González-Aparicio *et al. Hum. Cent. Comput. Inf. Sci.* (2017) 7:7
DOI 10.1186/s13673-017-0088-3

Human-centric Computing and Information Sciences

RESEARCH Open Access

Transaction processing in consistency-aware user's applications deployed on NoSQL databases

María Teresa González-Aparicio[1], Adewole Ogunyadeka[2], Muhammad Younas[2*], Javier Tuya[1] and Rubén Casado[3]

*Correspondence: m.younas@brookes.ac.uk
[2] Department of Computing and Communication. Technologies, Oxford Brookes University, Oxford OX33 1HX, UK
Full list of author information is available at the end of the article

Abstract

NoSQL databases are capable of storing and processing big data which is characterized by various properties such as volume, variety and velocity. Such databases are used in a variety of user applications that need large volume of data which is highly available and efficiently accessible. But they do not enforce or require strong data consistency nor do they support transactions. This paper investigates into the transaction processing in consistency-aware applications hosted on MongoDB and Riak which are two representatives of Document and Key-Value NoSQL databases, respectively. It develops new transaction schemes in order to provide NoSQL databases with transactional facilities as well as to analyze the effects of transactions on data consistency and efficiency in user's applications. The proposed schemes are evaluated using the YCSB + T benchmark which is based on Yahoo! Cloud Services Benchmark (YCSB). Experimental results show that using the proposed schemes, strong consistency can be achieved in MongoDB and Riak without severely affecting their efficiency. We also conduct experiments in order to analyse the level of consistencyof MongoDB and Riak transactional systems.

Keywords: NoSQL, Document databases, Key-value databases, Transactions, Consistency, Availability, Efficiency

Background

Data storage and processing needs of modern Internet services such as social networks, online shopping, data analytics and visualization have necessitated new kind of storage systems, called NoSQL (Not Only SQL) databases. Such databases use new techniques that support parallel processing and replication of data across multiple nodes in order to ensure improved performance and availability of data [1].

Unlike relational databases [2], NoSQL databases process and manage big data which is characterised by 3Vs (Volume, Variety, Velocity) [3]. NoSQL databases are required to support a variety of applications that need different levels of performance, consistency, availability and scalability [4]. For example, social media such as Twitter and Facebook [5] generate terabytes of daily data which is beyond the processing capabilities of

relational databases. Such applications need high performance but may not need strong consistency.

Different vendors design and implement NoSQL databases differently. Indeed, there are different types of NoSQL databases such as document databases, key-value databases, column stores and graph databases. But their common objective is to use data replication in order to ensure high efficiency, availability and scalability of data. Majority of NoSQL databases support eventual consistency instead of strong consistency. They do not support database transactions which ensure strong data consistency. Eventual consistency guarantees that all updates will reach all replicas after a certain delay. It works for certain applications such as social media, advertisement records, etc. But some of the user's applications need strong data consistency. That's, bank account data must be consistent whenever any updates are made to data. In other applications, such as online multiplayer gaming applications usually store profile data of a large number of users that require strong consistency. Therefore, NoSQL databases would be useful for managing data in such applications. However, the lack of support for transactions, table joins and referential integrity in NoSQL databases, mean that they are not suitable for applications such as banking, online gaming, etc. Such applications require all data replicas should be made instantly consistent and applications should have the latest version of the data in the case of any updates.

The work presented in this paper is built on our ongoing project on NoSQL databases [6, 7]. This paper investigates into the transaction processing of the two representatives and widely used NoSQL Document and Key-value databases; MongoDB and Riak. MongoDB is used by various industries and organizations such as Bosch, CERN, eBay, Expedia, etc. Similarly, Riak is used by Best Buy, NHS (U.K.), ShopKeep and so on. MongoDB and Riak follow different models and design principles. The former follows document database model while the latter is a key-value database.

We propose new transaction models (or schemes) in order to provide NoSQL databases with transactional aspects as well as to analyze the effects of transactions on data consistency and efficiency. Implementing transactions in NoSQL databases is not trivial. The 3Vs characteristics of NoSQL databases make it difficult to apply transaction models. Similarly, the distinct underlying models and design principles significantly complicate the process of transaction processing in NoSQL databases. Building transaction models and implementing them involve various factors. Transactions must adhere to certain transaction properties or correctness criteria such as Atomicity, Consistency, Isolation, Durability (ACID), semantic atomicity, eventual consistency and so on [6]. They should cater for concurrent execution of requests accessing and modifying shared data. Locking, optimistic protocols, and Snapshot Isolation are some of the possible techniques used to ensure concurrency in databases. It is also required to provide ways for recovering transactions from failures in order to maintain the correctness of applications and data, for example, using forward or backward recovery mechanisms. Our work focuses on implementing transaction properties and the concurrency aspects. Recovery is crucial to transaction processing but it is beyond the scope of this paper.

The main contributions of this paper are described as follows.

- It examines the characteristics, types, models, and architectures of NoSQL databases. The objective is to offer an insight into the issues that impede transaction processing in NoSQL databases.
- It develops new systems in order to implement transactions in two different NoSQL databases, MongoDB and Riak, which are widely used in the industry. The new systems provide such databases with stronger consistency.
- It evaluates the proposed transaction schemes using the YCSB+T benchmark [8] which is based on Yahoo! Cloud Services Benchmark (YCSB). Evaluation provides interesting results and observations:
- The proposed systems ensure strong consistency in MongoDB and Riak.
- Explore the effect of strong consistency on the efficiency in MongoDB and Riak.
- Analyse the level of consistency in two MongoDB and Riak.

The remainder of the paper is structured as follows. "Analysis of NoSQL databases" section presents an analysis of the NoSQL databases and identifies issues related to transaction processing in such databases. "Related work" section reviews related work. "The transaction model and properties" section presents the proposed transaction model and transaction properties. "Transactional system for MongoDB" and "Transactional system for Riak" sections respectively describe the implementation of transactions in MongoDB and Riak. "User application scenarios" section describes the evaluation process and results. "Evaluation benchmark and experiments" section concludes the paper.

Analysis of NoSQL databases

NoSQL databases are designed in order to store, process and manage big data which is generally characterised by 3Vs (Volume, Variety, Velocity). *Volume* refers to the amount of data being stored and processed. NoSQL big data is always growing and is very large in size, for example, in Petabytes and Exabyte. Such data is replicated across different nodes and is processed in parallel. *Variety* refers to the various kinds of data structures such as structured, semi-structured and unstructured data. NoSQL databases do not have (rigid) schema structures and can handle de-normalized data. *Velocity* represents the speed at which data is generated (from different sources) and is captured or processed by the NoSQL databases.

Based on the characteristics of data and applications, various kinds of NoSQL databases have been developed. Different NoSQL databases follow different data models and architectures.

Data models and types of NoSQL databases

Depending on the type of application requirements, different types of NoSQL databases implement the 3Vs characteristics differently. Unlike relational databases, which are built on sound mathematical and theoretical model [2], NoSQL databases do not have standard or widely accepted model. Despite the differences in the design of different NoSQL databases, they all have simple data model and do not support transactions. They lack support for complex queries such as *joins across tables*. While relational databases rely heavily on normalization and referential integrity, NoSQL databases are not

strictly normalized. Generally, NoSQL databases do not implement multi-key transactions. A multi-key transaction is an operation that involves multiple data items, which are atomically grouped and processed in a single operation. Most NoSQL databases offer simple put and get operations which usually involve a single key operation. The most common types of NoSQL databases are as follows:

- Documents databases: These databases typically store data in a JSON-like (JavaScript Object Notation) structure. Each document contains one or more fields. Each document has an associated key and documents are allowed to have different field attributes. Examples of document databases include, MongoDB and CouchDB.
- Key-value databases: These are the most basic form of NoSQL databases where each record is stored as a key-value pair with the value being opaque to the system. As such, each record can only be queried by the key which ensures fast access to keys. Examples of key-value databases are: Riak, DynamoDB [9], Voldemort and Redis.
- Column stores: These store and manage structured and semi-structured data that can scale to very large number of nodes. Unlike the key-value systems, each record can accommodate multiple columns (attributes) and columns can be grouped together to form column families. Examples are HBase, BigTable and Cassandra.
- Graph databases: These represent relationships between objects in a graph structure. Nodes represent objects while edges represent relationships. Graph databases are useful when relationships between data objects are possibly more important than data. Examples are Neo4j and Giraph.

The work presented in this paper focuses on the MongoDB and Riak which respectively belong to the Document and Key-Value classes of NoSQL databases. Table 1 presents a brief summary of the similarities and differences between MongoDB and Riak. Table 1 shows that MongoDB and Riak do not support transactions nor do they support normalization. They both are schema free and follow the eventual consistency model.

Architecture of NoSQL databases

NoSQL databases generally follow loosely coupled architecture [10] instead of tightly-coupled architecture as in relational databases. This allows NoSQL databases to separate system state from application state and to achieve high efficiency and scalability [11]. For example, MongoDB [12] architecture includes configuration server, router and database

Table 1 Main features of MongoDB and Riak

Features	Riak	MongoDB
Model	Key-value	Document
Replication	Peer-to-peer	Master–slave
Data schema	Schema-free	Schema-free
Standard data types	No	Yes
Consistency	Eventual	Eventual
Transactions	No	No
Normalization	No	No
Sharding	No	Yes

servers. The configuration server manages meta-data and stores information on data location in a cluster, while the router intercepts queries and directs them to appropriate nodes. Google system follows loosely coupled architecture which consists of Google File System (File tier) [13], Bigtable (record manager) [14], and Megastore (for transaction) [15]. Other examples of key-value databases, following loosely coupled architecture, are Spanner [16], G-Store [17], Deuteromony [18], CloudTPS [19], and ReTSO [6, 20].

Research issues

The underlying data models make it difficult to implement transactions in NoSQL databases. The main issues, pertaining to transaction processing, are as follows:

- *The adoption of simple* data *model with little or no normalization of data:* NoSQL data do not follow standard principles of normalization. This is to simplify data partitioning across multiple nodes into various shards. The partitioned data tends to have a shard key which determines the placement of data. However, de-normalised data may result in inconsistency and lack of integrity among data entities.
- *Replication of data:* In NoSQL, data are replicated over multiple nodes in order to provide high availability and efficiency. However, replication complicates the process of ensuring data consistency when data are concurrently read/updated by multiple requests (or transactions).
- *Lack of support for join operations and the inadequacy for formulating complex queries:* The de-normalizing and flattening out data into a single table imply that join operations are not required. Though this simplifies query processing, it lacks the power and flexibility of designing useful queries which are available in relational databases.
- *Relaxation of consistency and integrity and lack of multi-key (and cross tables) transactions:* NoSQL databases support relaxed (or eventual) consistency. Strong consistency can be achieved when data is manipulated by an individual (or single) operation. Consistency and integrity are not supported when multiple (or group of) operations manipulate shared data.

Related work

The original models of NoSQL databases do not support transactions, as described above. In order to address this issue, various transactional systems have been emerged for NoSQL databases. Depending on the data models and architectural styles existing approaches handle transactions at three different levels such as middleware, data store, and client side. Examples of middleware transactional systems include, Deuteromony [18], G-Store [17], Google Megastore [15], and CloudTPS [19]. Systems such as Spanner [16], and Granola [21] handle transactions at the data store level. ReTSO [20] handle transactions at the client side.

Unlike our approach, existing transactional systems neither implement semantic atomicity nor context-aware consistency. In addition, they do not support multi-key transactions the way we have implemented them in the NoSQL database. For instance, based on middleware approach, Deuteronomy [18] separates the transactional component from the data component. In contrast to the approach proposed in this paper,

Deuteronomy makes use of locking mechanism to manage concurrency and ensure consistency. Locking is useful but it has negative effects on the performance of transactions.

G-Store [17] proposes a key grouping protocol to group keys for applications that need multi-key transactions. Transactions are limited to within a group and G-Store cannot provide transactions across groups. Megastore [15], uses entity groups formation similar to G-store. But in Megastore, group formation is static and an entity belongs to a single group throughout the life span of that entity. As such, ACID transactions can only take place within specified groups.

CloudTPS [19], like Deuteronomy, make use of two layers architecture which includes LTM (Local Transaction Manager) and the cloud storage. Transactions are replicated across LTMs to preserve consistency in the presence of failures. ReTSO [20], similar to this system, makes use of Snapshot Isolation to ensure consistency. However, ReTSO is limited to single-key transactions and does not support multi-key transactions.

Various models and techniques have been developed in order to evaluate the performance of MongoDB in comparison with relational as well as NoSQL databases. For instance, experiments in [22] show a comparison between the performance of MongoDB and a well-known relational database, SQL Server. The results show that MongoDB runs faster for most operations including 'inserts' and 'updates'. However, SQL server performed better for more complex operations on non-indexed attributes. In [23], the authors compare the performance of MongoDB with the performance of Cassandra (a well-known column-oriented NoSQL database). The experiments show that MongoDB has a lower processing time than Cassandra for 'read' operations while Cassandra performs better when executing 'write' operations. However, the experiments carried out in [24] show slightly different results when the performance of MongoDB is compared with Hadoop. They show that Hadoop outperforms MongoDB in terms of performance latency for both 'read' and 'write' operations.

Furthermore, Riak has also been analysed in different research studies [25–27], by taking into account various factors such as the trade-off between CAP properties (Consistency, Availability, Partition tolerance) and performance. For instance, the work in [25] compares Cassandra, MongoDB and Riak. This comparative evaluation shows that generally Cassandra performs better for read-only, write-only and read/write operations than the MongoDB and Riak. However, compared to MongoDB, Riak shows better performance. Moreover, as the level of consistency increases then the number of operations per second (or throughput) decrease in Cassandra and Riak, for example, 25% decrease happens in Cassandra and 10% decrease happens in Riak. Authors in [26] studied the latency of databases, which is measured as the time required to perform a specific operation over a database. If the number of operations is very high, it means that the database starts being saturated. Therefore, it has a negative impact on the execution latency of database operations. Though, studies like [25] and [26] provide useful insights into the comparison of NoSQL databases various other factors, such as database architectures, data models and query capabilities, could have been taken into account in the evaluation process [27].

The transaction model and properties

This section describes the fundamental concepts, definitions and properties of transactions which are implemented in the proposed transaction processing systems.

NoSQL transactions

A NoSQL transaction (denoted as *NST*) is defined as the execution of an application which comprises a sequence of operations that provide transitions between (semantically) consistent states of the NoSQL data.

Definition 1 A *NST* can be formally defined as a tuple, $NST = (OP, PaO)$, where *OP* is a set of operations, $OP = \{OP_i \mid i = 1...n\}$, and *PaO* is a partial ordering of the operations which determines their order of execution. For instance, $OP_i > OP_j$ represents that OP_i executes before OP_j.

$OP_i^r[DE]$ represents a read operation of *NST*; meaning that *NST* reads a data entity, *DE*, from a NoSQL database. $OP_i^w[DE]$ represents a write operation of *NST*; meaning that *NST* writes (updates) a data entity, *DE*, to a NoSQL database. Such operations ($OP_i^r[DE]$ and $OP_i^w[DE]$) are used to model the CRUD (Create, Read, Update and Delete) operations which are most commonly implemented in NoSQL systems such as Riak and MongoDB.

Further, $OP_i^r[DE]$ represents the Read (of CRUD) and $OP_i^w[DE]$ represents the Create, Update or Delete operation (of CRUD). $OP_i^r[DE]$ reads data without any modification to the data, *DE*. $OP_i^w[DE]$ writes data *DE*, i.e., data can be modified through Create, Update or Delete operation.

In addition, to data read/write operations, *NST* is also associated with (control) operations which include: begin or start, commit and abort. These are explained as follow.

Begin or start operation

The execution of each *NST* must be marked through *begin* or *start* operation. That is, *NST* should begin first before any of its operations ($OP = \{OP_i \mid i = 1...n\} \in NST$) can be executed.

Commit and abort operations

Each *NST* terminates with either a commit or an abort operation. If *NST* is successfully executed then it terminates with a 'commit' operation. If *NST* cannot be successfully executed then it terminates with an 'abort' operation.

For example, *NST* for a banking transaction (as in YCSB + T benchmark [8]) comprises a sequence of operations. After starting execution (*Begin*) it can read records ($OP_i^r[DE]$) of bank accounts, deducts money ($OP_i^w[DE]$) from one account and deposits ($OP_i^w[DE]$) that money into another account. If all these operations are successful then *NST* is committed (*Cmt*). Otherwise, *NST* is aborted (*Abt*).

Properties of transactions

Each *NST* is required to follow certain properties. The properties adopted in the proposed transaction models are explained as follows.

Atomicity requires that all the operations of a transaction must successfully execute or not at all. In other words, a transaction is considered as an atomic unit of work. *Semantic Atomicity* allows flexibility in that depending on semantics (requirements) of a transaction, some operations must execute successfully while others may not. *Consistency* states that data must be consistent before and after the execution of a transaction. That is, any updates made to data by a transaction should be consistently (and instantly) reflected in the database. *Eventual* [28, 29] or flexible consistency requires that all updates will reach all replicas after a certain delay. Some of the replicas might be inconsistent during the 'delay' period. *Isolation* requires that transactions must not expose their intermediate results to other transactions which are concurrently running and possibly sharing data. *Durability* requires that results of a completed transaction must be made permanent in a database so as to provide recovery in the event of failures.

Durability is supported by almost all NoSQL databases. But majority of them do not follow atomicity or semantic atomicity (of multiple operations) as they do not support transactions. In terms of isolation and consistency, different NoSQL databases adjust the level of consistency differently. The parameters, N, W and R are generally used to adjust the level of consistency with respect to the efficiency of read and write operations [6]. 'N' represents the number of replicas of each data item that the NoSQL database will maintain. 'R' represents the number of replicas that the application will access when reading the data item. 'W' represents the number of replicas of the data item that must be written before the write can complete.

In order to ensure strong consistency, $N = W$; that is, all replicas of the data item need to be updated (written) before the data item can be accessed by an application. However such level of consistency results in poor efficiency as all replicas need to be synchronized. Weaker level of consistency can be achieved by setting $W < N$, such as $W = 1$ and $N = 3$ (assume that the number of replicas is 3). In this case, write operation has to update one replica of the data item. Other replicas can be updated after certain delay and made consistent (which is the case of eventual consistency).

Note that in the current NoSQL databases, strong consistency (with $N = W$) is ensured when data is manipulated by a single CRUD operation, e.g., $OP_i^w[DE]$. However, strong consistency is not supported when multiple CRUD operations are executed together as a transaction. For instance, when executing multiple operations $OP_i^r[DE]$, $OP_i^w[DE]$, $OP_i^w[DE]$ as a part of transaction, *NST*. This is because they do not support transactions.

In the following we develop new transaction systems that enforce the transaction properties in NoSQL databases MongoDB and Riak.

Transactional system for MongoDB

We implement the transaction model in the MongoDB in such a way that fully enforces the ACID properties and with a strong consistency (i.e., $N = W$). However, implementing ACID transactions is complex as MongoDB does not provide support for such transactions. Our approach is to implement transactions as Multi-Key transactions [7] using the Snapshot Isolation technique [30]. Snapshot Isolation is an optimistic concurrency control technique which allows for higher concurrency. A multi-key transaction is an execution of a cloud application that involves multiple data key items and comprises

different operations. A multi-key transaction provides transitions between consistent states of the shared data and follows the ACID properties of transactions.

The proposed transaction system follows loosely coupled architecture in order to separate the implementation of transactional logic from the underlying data structure of MongoDB. Figure 1 represents the architecture which comprises three main components: Transaction Processing Engine, Data Management Store, and Times Stamp Manager. Due to space limitation, we briefly explain the main functions of each of these components.

Transaction processing engine (TPE)

The TPE is responsible for implementing Multi-Key transactions in the system. In order to run NoSQL operations ($OP_i^r[DE]$, $OP_i^w[DE]$) as a transaction, clients (applications) submit their requests to the TPE. TPE receives such requests from clients and manage them as transactions. TPE also maintains data schema which is required for transaction processing and management. Note that MongoDB is a schema free database. TPE also defines relationships between different data entities (*DE*). This is to enforce integrity constraints and strong consistency. Based on the relationships between data entities, TPE facilitates *join operations* which are not implicitly supported in MongoDB. The ability for the TPE to facilitate join expressions means that complex operations can be implemented by the system and these operations can be implemented atomically, i.e., with transactional semantics. This feature is absent from existing NoSQL databases.

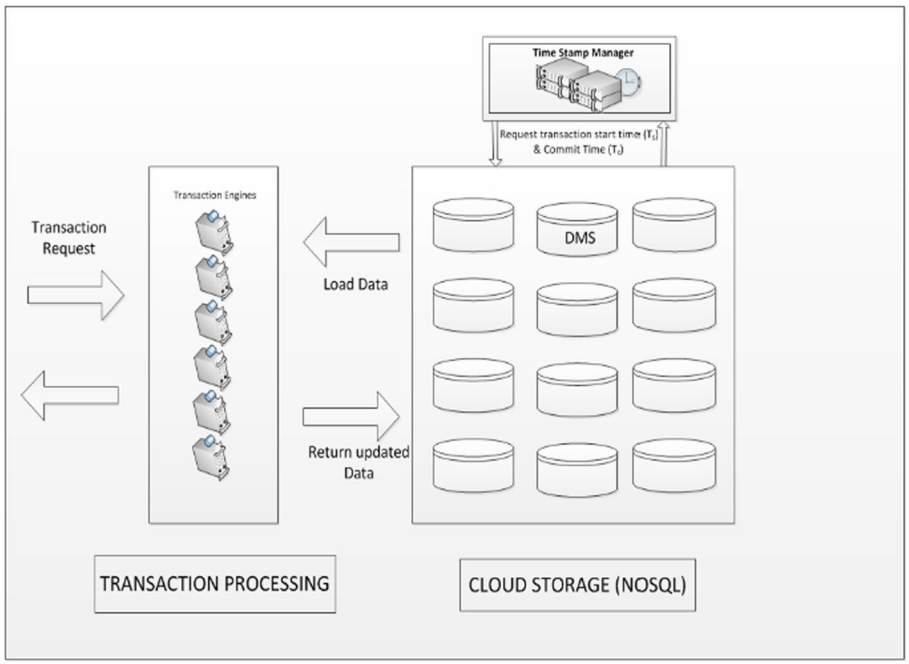

Fig. 1 Architecture of the proposed system for MongoDB

Data management store (DMS)

This component is implemented as part of the MongoDB. DMS stores all the data persistently. This component is highly scalable in order to meet big data storage requirements. It replicates data in terms of different replicas in order to ensure improved efficiency, high availability and fault tolerance. Further, the DMS implements the Snapshot Isolation protocol in cooperation with the Time Stamp Manager (TSM) in order to provide concurrency of transactions.

Times stamp manager (TSM)

The TSM manages the ordering and scheduling of transactions in the proposed system. It interacts with DSM and TPE in order to schedule the execution of the different operations (*Begin,* $OP_i^r[DE]$, $OP_i^w[DE]$, *Cmt* and *Abt*) of a transaction. The objective is to maintain consistency of data when concurrently manipulated by different transactions. Concurrency is provided through the implementation of the Snapshot Isolation technique. This is a non-blocking protocol and provides higher concurrency and high efficiency in transactions processing.

Based on the above implementation of the transaction system we conduct various experiments which are described in "Evaluation benchmark and experiments" section.

Transactional system for Riak

Transactional system for Riak is implemented differently than that of the MongoDB. This is due to the differences in the data models and types of MongoDB and Riak as highlighted in Table 1 above. Transactional system for Riak implements semantic atomicity and flexible or context-aware consistency. In other words, depending on the semantics and contextual information different applications are provided with different levels of atomicity and consistency.

Note that the focus of this paper is not to compare the implementation of transactions in Riak with MongoDB. Rather the paper aims to compare the effect of implementing transactions in individual NoSQL databases of Riak and MongoDB.

Figure 2 represents the architecture and the main components of the proposed system. It is based on a loosely coupled architecture in order to separate the implementation of transactional logic from the underlying data structure of Riak. The main components described as follows.

Coordinator

Coordinator manages the overall execution of CRUD operations ($OP_i^r[DE]$, $OP_i^w[DE]$). It ensures that every CRUD operation is executed in such a way that the outcome of the whole operation is correct and that the desired context is also fulfilled. It interacts with the 'Data and Semantic Rules', which contains information about data design and related semantic rules. This module provides information about the different types of data the application should handle and the relationships between them. Semantic rules are used to establish and identify relationships between data entities. These are also used to help in establishing semantic atomicity of CRUD operations when executed as a transaction.

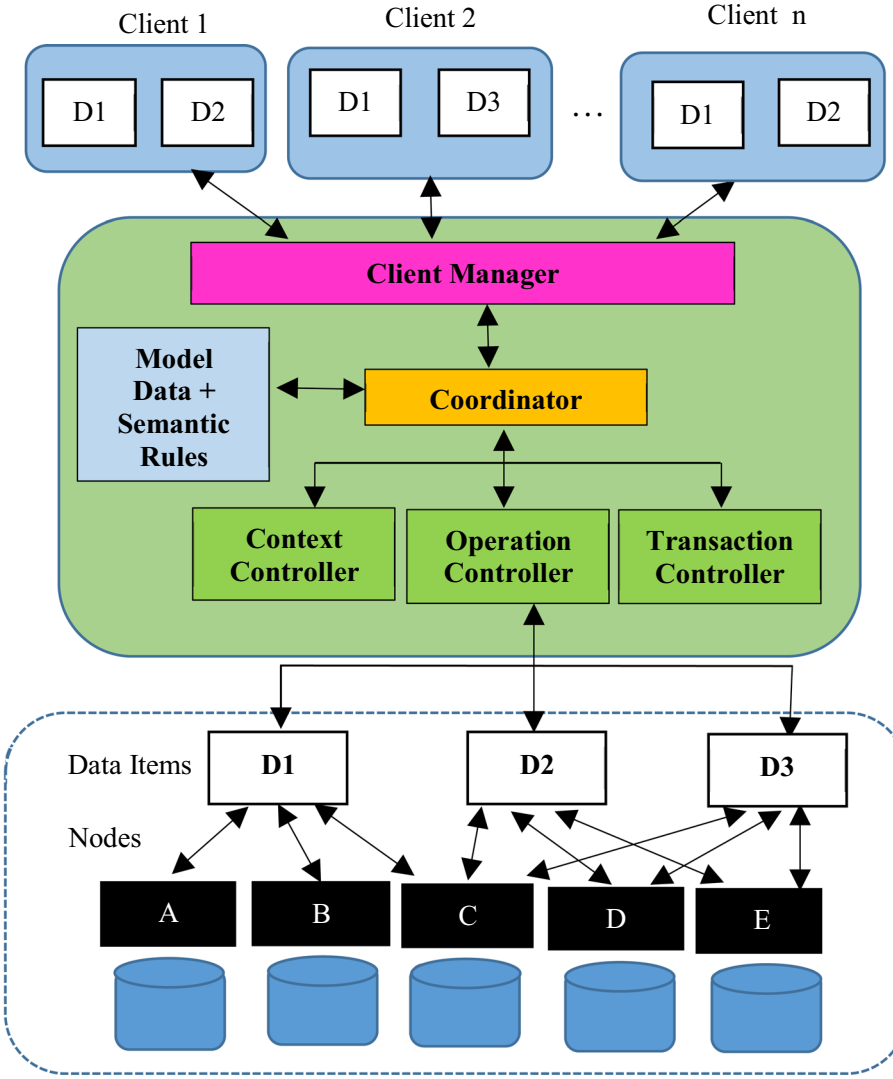

Fig. 2 Architecture of the proposed system for Riak

Context controller

This deals with managing contextual information related to the execution of CRUD operations and the data. We define two classes of context information: system context and client (or application) context. System context information can come from (Riak) system configuration—i.e., the way it is configured to execute CRUD operations and to manage the data internally, such as configuring the level of consistency using N, W, R parameters; creating and processing different replicas of the same data; and the distribution of replicas among different nodes. Client context information is related to the application needs such as the level of consistency (they require), required response time, and the level of atomicity (semantic atomicity).

Operation controller

It provides an interface or interaction between the Coordinator and the Riak database which is represented as the lower layer in Fig. 2. It enables communication with Riak when CRUD operations are executed by the Coordinator. It also provides Coordinator with information about the states (existing and new states) of the Riak database.

Transaction controller

In collaboration with the above modules, Transaction Controller deals with the transactional features of the CRUD operations—i.e., CRUD operations can be successfully executed if it meets the required context and transactional properties. If not, then the operations have to be rolled back in order to maintain the required level of consistency. For instance, a banking transaction (as described above) requires strong consistency and atomicity. Therefore, all the operations of a banking transaction should be successfully completed in order to ensure consistency of data across bank accounts.

User application scenarios

This section explains the implementation of an online multi-player gaming application hosted on both MongoDB and Riak NoSQL databases. The implementation is provided as a proof of concept. It is based on the architecture of both MongoDB and Riak, which has been described in "Transactional System for MongoDB" and "Transactional system for Riak" sections respectively (see Figs. 1, 2).

MongoDB implementation

In the MongoDB implementation, the user data (user profiles, attributes and gaming information) is stored in the DMS layer. Recall that the DMS is a highly scalable component and stores data persistently for the system. This allows the system to be able to effectively manage user information for high number of users. In MongoDB, players profile can be stored as a document and each user/player is identified by the document ID.

A game is played by multiple users who come together to start a new game. The ID and profile of each of these users are sent to the TPE. During the progress of a game, changes to player information should be transactional to maintain consistency of the ongoing game. Recall that the TPE manages transactions for the system and the TPE stores schema information. Therefore, the TPE is in charge of implementing all user interactions, transactional semantics and applications schema information. The TPE interacts with the TSM to manage ordering and scheduling of user requests during a game. This will help to manage the consistency of the gaming application. Information that transcends a game (such as player profile or player ratings) is then stored back at the DMS for future games. Note that the implementation is used as a proof of concept and does not provide full implementation of a gaming application.

Riak implementation

In Riak implementation, the coordinator supervises the whole process of different modules involved in processing a transaction. In the prototype implementation, operations are treated as transactions in order to guarantee consistency of information such as user ID and profile. For this reason, every operation or a set of operations are controlled

by a module named as "Transaction Controller". During this process, (user) data are retrieved, stored or removed from/to the database by the module "Operation Controller" in relation to the game status and player's profile management. At the end, player's profile must be in consistent with the final results.

Evaluation benchmark and experiments

This section presents the evaluation benchmark and the experimental results in order to evaluate the transactional systems of Riak and MongoDB. During this process, data concurrency has been implemented with Snapshot Isolation in these databases. Moreover, each of them has been developed with a different architecture ("Transactional System for MongoDB" and "Transactional system for Riak" sections) and implementation ("Evaluation of the MongoDB transactional system" and "Evaluation of the riak transactional system" sections).

Evaluation benchmark

We adopt the YCSB+T benchmark [8], which is an extension of the Yahoo! Cloud Services Benchmark (YCSB) benchmark. YCSB+T takes into account transactions. But YCSB is applicable to evaluating single NoSQL operation rather than a group of operations (as in transactions).

We use the closed economy workload of the YCSB+T benchmark, in order to quantify database anomalies. This provides an environment for executing multiple concurrent transactions on NoSQL data. Transactions can perform conflicting operations on shared data, e.g., $OP_i^r[DE]$ and $OP_i^w[DE]$ concurrently access same data entity, *DE*. This may introduce in data inconsistency.

Evaluation of the MongoDB transactional system

The MongoDB transactional system is implemented using Python language and MongoDB was used as a data store. The implementation is carried out on a 16 GB RAM PC and Windows OS. The following experiment is conducted in order to evaluate the efficiency of the transactions and their impact on ensuring strong consistency.

Experiment 1

This experiment evaluates the efficiency of transactions and the consistency by taking into account the number of transactions and the time taken to process such transactions.

The number of transactions per client is 100. The number of clients varies from 1 to 32. That is, with one client the system executes 100 transactions (per sec) and with 32 clients the system executes 3200 transactions (per sec). The results are shown in Table 2 and Fig. 3. These take into account transactional and non-transactional (simple CRUD) operations. It considers the configuration of MongoDB where the number of replicas is 3 (i.e., $N = 3$). As described above (see "Properties of transactions" section), the different NoSQL databases assign different values to the parameters, N, W and R in order to adjust the level of consistency with respect to the efficiency of read and write operations [6].

The column 'No of Clients' in Table 2 shows the number of clients concurrently submitting transaction requests to the system. 'Update only' column shows the efficiency of

Table 2 Efficiency of transactions in MongoDB

No of clients	Update only (sec.)	Transactions (sec.)
1	0.008	0.027
2	0.006	0.021
4	0.005	0.021
8	0.005	0.022
16	0.006	0.059
32	0.007	0.04

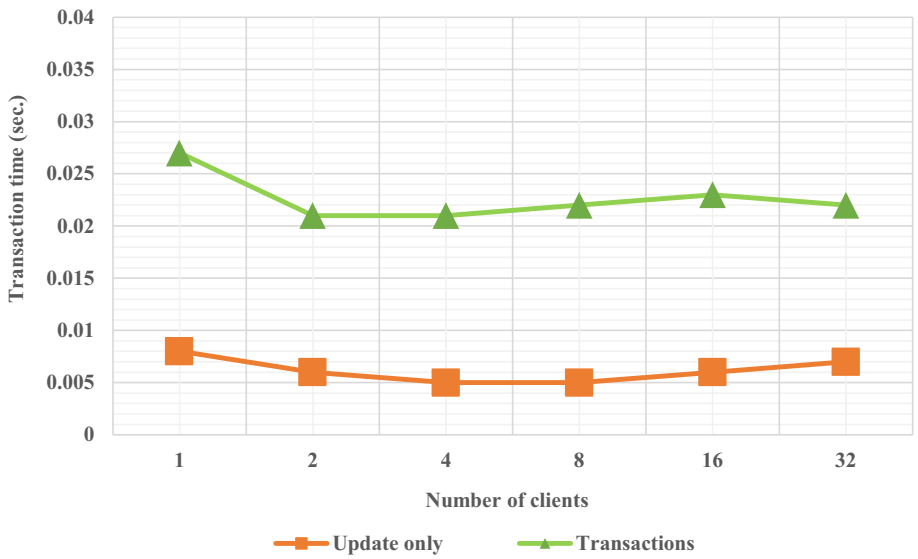

Fig. 3 Execution time per transaction in MongoDB

the operations without transactions. In other words, the level of consistency is weaker. The column 'Transactions' shows the efficiency of transactions with strong consistency (where W = N=3; the number of replicas is 3). As shown in the results that transactions incur extra processing time compared to simple 'Update only' (or Write) operations but they enforce 100% consistency. 'Update only' operation can update data but without transactions. In other words, transaction properties such as atomicity, consistency, etc. are not enforced. The processing overhead of transactions is incurred due to the coordination of transaction operations on the data. However, this is to consistently reflect updates in the different replicas of the shared data.

Evaluation of the Riak transactional system

The Riak transactional system is implemented with Oracle Java 7 as the programming language, and the NoSQL key/value Riak (by Basho) 2.1.1 as a database. The experiment has been conducted using the following hardware/software features: a CPU core with 2.4 GHz Intel(R) Core(TM) i7-5500, the operating system Ubuntu 14.04 LTS of 64 bits.

Experiment 2

We take the same measures as in Experiment 1. That is, the number of transactions per client is 100. The number of clients varies from 1 to 32. Again, with one client the system executes 100 transactions (per sec) and with 32 clients the system executes 3200 transactions (per sec).

The results are shown in Table 3 and Fig. 4. 'Update only' and 'Transactions' columns represent the same information as in experiment 1. 'Update only' operations update data but without following transactions rules (properties). Using 'Transactions' data is read and update by following the rules of transaction properties (see "Properties of transactions" section).

As shown in the results, transactions incur extra processing time compared to simple 'Update only' (or Write) operations but they enforce 100% consistency. If the number of transactions and clients is smaller (2 clients) then transactions do not incur much processing overhead. However, when the number of clients increases from 16 to 32, then the transaction execution time also increases. This is due to the fact that transactions manipulate shared data which generate conflicts between different transactions. Similar to MongoDB, the processing overhead of transactions is incurred due to the coordination of transaction operations on the data.

Table 3 Efficiency of transactions in Riak

No of clients	Update only (sec.)	Transactions (sec.)
1	0.012	0.03
2	0.018	0.034
4	0.028	0.05
8	0.05	0.085
16	0.094	0.159
32	0.176	0.348

Fig. 4 Execution time per transaction in Riak

Analysis of the data consistency in MongoDB and Riak

We evaluated the level of consistency in MongoDB and Riak. That is, the consistency of the NoSQL database (MongoDB and Riak) should be guaranteed every time a transaction is executed. We evaluate the consistency by including 'read' operations which read shared data that is possibly updated by ongoing (running) transactions. The experimental results are shown in Fig. 5. The figure shows the number of 'read' operations that may read inconsistent data—i.e., data which is updated by transactions. In other words, the data may or may not be correct as it is read from an ongoing transaction. The experiment shows that if the numbers of clients ranges from 1 to 4 and the number of transactions ranges from 100 to 400 then there is not much difference in the level of consistency between MongoDB and Riak. However, if the numbers of clients increases to 8 (or above) then Riak tends to ensure better consistency than MongoDB.

As shown in Fig. 5, there is a noticeable difference in the number of read consistencies (#total reads vs #read inconsistencies) between MongoDB and Riak when the number of clients increases above 16. Firstly, in MongoDB, the number of read consistencies is 468 (1600–1132) and 531 (3200–2669) for 16 and 32 clients respectively. Secondly, in Riak the number of read consistencies is 648 (1600–952) and 812 (3200–2388) for 16 and 32 clients respectively. This fact implies that the number of read consistencies approximately increase 1.13 (531/468) and 1.25 (812/648) times when number of clients in the system fluctuate from 16 to 32, in MongoDB and Riak respectively.

From another point of view, it is highly likely an increase in the percentage of read inconsistencies in both systems as the number of concurrent requests grow, MongoDB (16 clients: 71% [1132/1600*100]; 32 clients: 83% [2669/3200*100]) and Riak (16 clients: 59% [952/1600*100]; 32 clients: 75% [2388/3200*100]). As a result, this fact highlights that both transactional systems cause different number of read consistencies (or read inconsistencies) states in the NoSQL databases. Therefore, these data could

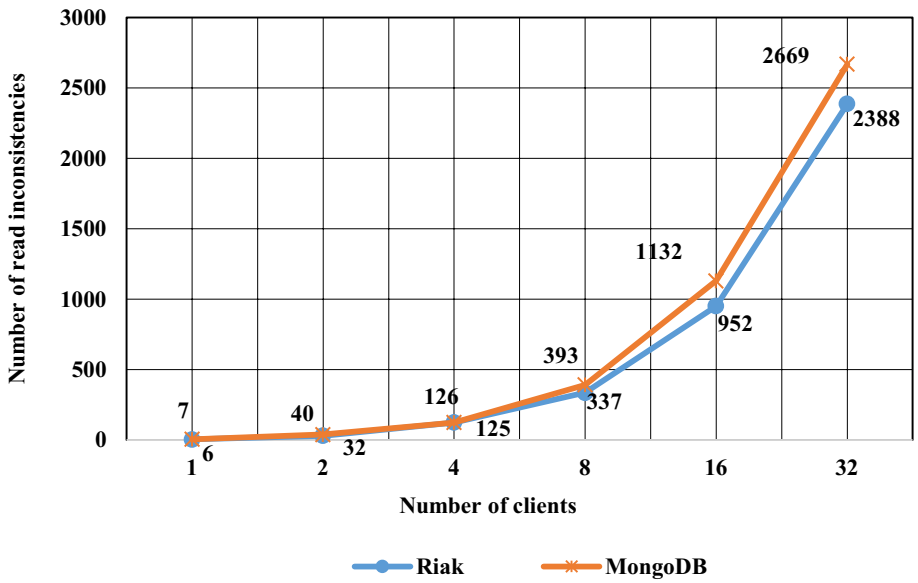

Fig. 5 Read inconsistencies

provide an insight in how much important a transactional management model (semantic atomicity+context-aware consistency with multi-key) would be.

Conclusion and future work

NoSQL databases play a prominent role in big data storage and processing and are used in various business, commercial and social applications such as Twitter, Facebook, Google mail, and Yahoo, to name a few. They have become champion of high efficiency and availability of big data but at a loss of strong data consistency, data normalization and transactions. In order to minimise the impact of such a loss, this paper addressed the issue of transactions and data consistency in NoSQL databases in general and MongoDB and Riak in particular.

In this paper we have examined the issues that impeded the transaction processing in NoSQL databases and have developed transactional systems for two distinct NoSQL databases, MongoDB and Riak. The transactional systems were implemented as prototype systems that process transactions, manage the concurrency of multiple transactions, and maintain the consistency of data stored in MongoDB and Riak. This paper also discussed the implementation and simulation of an online multi-player game application which uses Riak and MongoDB as data stores. The implementation follows the transaction model developed for the individual database. In each of the implementations, NoSQL databases (Riak and MongoDB) act as a stable storage for the user's application (i.e. application data is stored persistently in the NoSQL database). During this procedure, data in relation to player's profile should be kept in consistency according to the game's rules. For this reason, operations are treated as transactions in order to be performed or roll backed in case of rule violations. Moreover, the developed systems were evaluated using a transaction processing benchmark of YCSB+T. Various experiments were conducted in order to evaluate the efficiency and consistency of transaction processing in MongoDB and Riak.

The experiments provided some useful insights into the performance and level of consistency. It is observed that transaction processing systems incur performance overhead but they are crucial for NoSQL database applications that need strong consistency. Thus transactional guarantees (atomicity, consistency, etc.) do not need to be abandoned when implementing high performance applications in NoSQL databases. Focusing only on performance and compromising on consistency can lead to serious issues, for some applications, in NoSQL databases. One example is the Flexcoin (a Bitcoin exchange) that was closed down in March 2014 due to hacking. One of the issues was associated with the design of MongoDB which did not have sufficient mechanisms in place for ensuring consistency and concurrency [31].

Our future work plans to fully implement user's (gaming) application and to evaluate transaction systems using a real life big data set in the experimentation. We also plan to empirically evaluate the proposed systems in terms of consistency and performance in other user's applications that need strong consistency of data.

Authors' contributions

MT-GA and AO carried out related studies and analysis of the literature. MT-GA, AO, and MY participated in the design and development of the proposed schemes. MT-GA and AO carried out implementation and experiments. MY, JT and RC participated in the coordination and helped draft the manuscript. All authors read and approved the final manuscript.

Author details
[1] Department of Computing, University of Oviedo, Gijón, Spain. [2] Department of Computing and Communication. Technologies, Oxford Brookes University, Oxford OX33 1HX, UK. [3] Accenture Digital Spain, Madrid, Spain.

Acknowledgements
We would like to express our gratitude to the Oxford Brookes University, Oxford, UK, and two financial sponsors of this work: the Spanish Research, Development and Innovation Plan supported by the Ministry of Economy and Competitiveness (TIN2013-46928-C3-1-R and TIN2016-76956-C3-1-R) and the Principality of Asturias (GRUPIN14-007).

Competing interests
The authors declare that they have no competing interests.

Received: 2 September 2016 Accepted: 29 January 2017
Published online: 05 April 2017

References

1. DeWitt D, Gray J (1992) Parallel database systems: the future of high performance database systems. Commun ACM 35(6):85–98
2. Codd EF (1970) A relational model of data for large shared data banks. Commun ACM 13(6):377–387
3. Casado R, Younas M (2015) Emerging trends and technologies in big data processing. Concurr Comput 27(8):2078–2091
4. Abadi DJ (2012) Consistency tradeoffs in modern distributed database system design. Comput IEEE Comput Mag 45(2):37–42
5. Lakshman A, Malik P (2010) Cassandra: a decentralized structured storage system. ACM SIGOPS Oper Syst Rev 44(2):35–40
6. Gonzalez-Aparicio MT et al (2016) A new model for testing CRUD operations in a NoSQL databases. IEEE 30th international conference on advanced information networking and applications. Crans-Montana, Switzerland, pp 79–86
7. Ogunyadeka A et al (2016) A multi-key transactions model for NoSQL cloud database systems. In: IEEE second international conference on big data computing service and applications (BigDataService)
8. Dey A et al (2014) YCSB + T: benchmarking web-scale transactional databases. In: IEEE 30th international conference on data engineering workshops (ICDEW)
9. DeCandia G et al (2007) Dynamo: amazon's highly available key-value store. SIGOPS Oper Syst Rev. 41(6):205–220
10. Kraska T, Trushkowsky B (2013) The new database architectures. IEEE Internet Comput 17(3):72–75
11. Agrawal D et al (2010) Data Management Challenge. 6th international workshop on databases in networked information systems. Aizu-Wakamatsu, Japan, pp 1–10
12. MongoDB (2014) https://docs.mongodb.com/manual/. Accessed 5 Jan 2017
13. Ghemawat S, Gobioff H, Leung S-T (2003) The Google file system, in ACM SIGOPS operating systems. Review 37:29
14. Chang F et al (2008) Bigtable: a distributed storage system for structured data. ACM Trans Comput Syst 26(2):1–26
15. Baker J et al (2011) Megastore: providing scalable, highly available storage for interactive services. In: conference on innovative data systems research (CIDR), p 223–234
16. Corbett JC et al (2013) Spanner: Google's globally distributed database. ACM Trans Comput Syst. 31(3):1–22
17. Das S, Agrawal D, Abbadi AE (2010) G-Store: a scalable data store for transactional multi key access in the cloud. Proceedings of the 1st ACM symposium on cloud computing. Indianapolis, Indiana, pp 163–174
18. Levandoski JJ et al (2011) Deuteronomy: transaction support for cloud data. In: Conference on innovative data systems research (CIDR), p 123–133
19. Wei Z, Pierre G, Chi C-H (2012) CloudTPS: scalable transactions for Web applications in the cloud. IEEE Trans Serv Comput 5(4):525–539
20. Junqueira FB, Reed M, Yabandeh M (2011) Lock-free transactional support for large-scale storage systems. In: IEEE/IFIP 41st International conference on dependable systems and networks workshops (DSN-W), p 176–181
21. Cowling J, Liskov B (2012) Granola: low-overhead distributed transaction coordination, in USENIX ATC'12 Boston
22. Parker Z, Poe S, Vrbsky SV (2013) Comparing nosql mongodb to an sql db. In: Proceedings of the 51st ACM Southeast Conference
23. Abramova V, Bernardino J (2013) NoSQL databases: MongoDB vs cassandra. Proceedings of the International C* Conference on computer science and software engineering. ACM, New York, pp 14–22
24. Dede E et al (2013) Performance evaluation of a mongodb and hadoop platform for scientific data analysis. In: Proceedings of the 4th ACM workshop on Scientific cloud computing
25. Klein J et al (2015) Performance evaluation of NoSQL databases: a case study. The 1st ACM international workshop on performance analysis of big data systems. ACM, Austin
26. Copie A, Fortis TF, Munteanu VI (2013) Determining the performance of the databases in the context of cloud governance. In: Eighth international conference on P2P, parallel, grid, cloud and internet computing (3PGCIC)
27. Gorton I, Klein J, Nurgaliev A (2015) Architecture knowledge for evaluating scalable databases, In: The 12th working IEEE/IFIP conference on software architecture (WICSA)
28. Vogels W (2009) Eventually consisten. Commun ACM 52(1):40–44
29. Lloyd W et al (2011) Don't settle for eventual: scalable causal consistency for wide-area storage with COPS. Proceedings of the twenty-third ACM symposium on operating systems principles. ACM, Cascais, pp 401–416
30. Berenson H et al (1995) A critique of ANSI SQL isolation levels. SIGMOD Rec 24(2):1–10
31. Sirer EG (2014) NoSQL meets bitcoin and brings down two exchanges: the story of flexcoin and poloniex, http://hackingdistributed.com/2014/04/06/another-one-bites-the-dust-flexcoin/. Accessed 26 Jan 2017